UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Corso di Laurea in Fisica

Tesi di Laurea

Deep Neural Networks for energy reconstruction of Inverse Beta Decay events in JUNO

Relatore

Prof. Alberto Garfagnini

Correlatore

Dr. Yury Malyshkin

Laureando

Francesco Manzali

Anno Accademico 2018/2019

Abstract

The Jiangmen Underground Neutrino Observatory (JUNO) is a scintillation detector, currently under construction, which aims to solve the neutrino mass hierarchy by measuring reactor $\overline{\nu}_e$ energy spectrum with a resolution of $3\%/\sqrt{\mathcal{E}(\mathrm{MeV})}$ - the highest ever achieved in a large mass neutrino detector. Several approaches for energy reconstruction are being evaluated on simulated data, and Deep Learning methods have already shown promising results, both in accuracy and efficiency. In this work, a new Convolutional Neural Network with a rotational invariant architecture is trained on a small dataset of 160k instances, and is fine-tuned to exploit the detector's spherical symmetry and make use of position and timing data from individual photomultipliers. This approach proves to be insensitive to the presence of dark noise from thermal fluctuations, leading to a $(2.45 \pm 0.03)\%$ visual energy resolution at 2 MeV, only slightly higher than the 2.2% expected from theory, with a reconstruction bias well below 1%. However, a simpler Fully Connected Neural Network, replicated from previous work, which uses only integral data and is trained on a larger dataset (750k instances), leads to a slightly better resolution of $(2.26 \pm 0.05)\%$ at 2 MeV, while being more sensitive to added noise - proving that there could still be some margin of improvement for more complex methods.

Organization

The first chapter explains the main specifics of JUNO, and the Inverse Beta Decay and scintillation detectors physics. The task of energy reconstruction is introduced in the second chapter, along with a quick review of supervised learning and Deep Neural Networks. In the third chapter two deep learning approaches are explored: a baseline model based upon previous work and a new Spherical Convolutional Neural Network with rotational invariance. Their performance is then evaluated on the available datasets, with or without the presence of noise. At last, conclusions are drawn in the final chapter.

This page is intentionally left blank.

Contents

Notation

Multidimensional quantities are denoted in bold: vectors with lowercase letters, higher order tensors with uppercase letters. The true value of observables used in simulations is denoted with a *. For example $\mathcal{E}^{*,(i)}$ is the true value of energy for the i-th event.
The following abbreviations will be used:

IBD	Inverse Beta Decay	**MAPE**	Mean Absolute Percentage Error
PE	Photoelectron	**MSE**	Mean Squared Error
FV	Fiducial Volume	**DNN**	Deep Neural Network
PMT	Photomultiplier	**CNN**	Convolutional Neural Network
DN	Dark Noise	**MLP**	MultiLayer Perceptron
MAE	Mean Absolute Error		

Chapter 1

Introduction

1.1 The JUNO Experiment

The Jiangmen Underground Neutrino Observatory [1] is a large scintillation detector currently being built in Jinji town near Kaiping city, within the Guangdong province in Southern China. Its main goal is to establish the *neutrino mass ordering*.
It is known from particle physics that neutrinos are produced in definite *flavors* - electronic ($|\nu_e\rangle$), muonic ($|\nu_\mu\rangle$) and tauonic ($|\nu_\tau\rangle$). However, experimental evidence, for example that of the 1998 Super-Kamiokande detector [2], shows that neutrinos travel through spacetime in mass eigenstates ($|\nu_1\rangle$, $|\nu_2\rangle$ and $|\nu_3\rangle$) which are distinct from the flavor eigenstates. The time evolution operator acts differently on the different masses m_i, and this leads to the phenomenon of *neutrino oscillation*, where a neutrino of initial flavor ν_i can be detected as a neutrino with flavor ν_f after a certain travelled distance L.
More precisely, the relation between mass eigenstates $\{|\nu_i\rangle\}$ and flavor eigenstates $\{|\nu_\alpha\rangle\}$ is given by the 3×3 Maki-Nakagawa-Sakata-Pontecorvo [3, 4, 5] (MNSP) matrix U:

$$\begin{pmatrix} \nu_e \\ \nu_\mu \\ \nu_\tau \end{pmatrix} = \begin{pmatrix} U_{e1} & U_{e2} & U_{e3} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} \\ U_{\tau 1} & U_{\tau 2} & U_{\tau 3} \end{pmatrix} \begin{pmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{pmatrix}$$

Assuming U is exactly unitary, it can be parametrized in a standard way using 3 *mixing angles* $\theta_{12}, \theta_{13}, \theta_{23}$ and a parameter δ called the *Dirac CP-violating phase*.

The main sources of neutrinos in the JUNO experiment are two nuclear power plants located at 53 km from the detector, which produce mainly anti-electron neutrinos ($\bar{\nu}_e$) with energies lower than 10 MeV [1, p. 188]. These neutrinos can't be flavor-tagged after oscillating to $\overline{\nu}_\mu$ or $\overline{\nu}_\tau$, as they do not meet the required threshold for generating a μ or τ. So, the most interesting observable is simply their survival probability:

$$P(\bar{\nu}_e \to \bar{\nu}_e) = 1 - \sin^2 2\theta_{12} c_{13}^4 \sin^2\left(\frac{\Delta m_{21}^2 L}{4\mathcal{E}}\right) - \sin^2 2\theta_{13}\left[c_{12}^2 \sin^2\left(\frac{\Delta m_{31}^2 L}{4\mathcal{E}}\right) + s_{12}^2 \sin^2\left(\frac{\Delta m_{32}^2 L}{4\mathcal{E}}\right)\right] \tag{1.1}$$

where $s_{ij} \equiv \sin\theta_{ij}$, $c_{ij} \equiv \cos\theta_{ij}$, $\mathcal{E}$ is the neutrino energy, L the travelled distance and $\Delta m_{ij}^2 \equiv m_i^2 - m_j^2$. Past experiments have already given estimates for Δm_{21}^2, $|\Delta m_{31}^2|$ and the 3 mixing angles.

The aim of JUNO is then to improve these results, and especially fix the sign of Δm^2_{31} by discriminating between two possibilities (fig. 1.1):

- **Normal Ordering** (NO), where $|\Delta m^2_{31}| = |\Delta m^2_{32}| + |\Delta m^2_{21}|$, and $m_1 < m_2 < m_3$
- **Inverted Ordering** (IO), where $|\Delta m^2_{31}| = |\Delta m^2_{32}| - |\Delta m^2_{21}|$, and $m_3 < m_1 < m_2$

In fact, depending on the sign of Δm^2_{31}, the plot of (1.1) is slightly different, as can be seen in fig. 1.2. The medium baseline available to JUNO is optimal for distinguishing the two possibilities, but still an exceptional energy resolution is required.

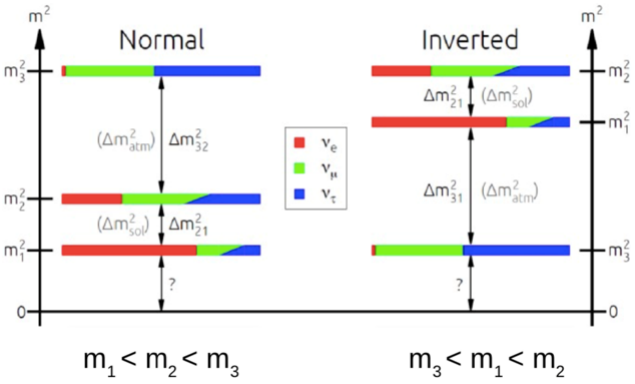

Figure 1.1 – Depending on the sign of Δm^2_{31}, two orderings are possible for the neutrino masses. Image from [6].

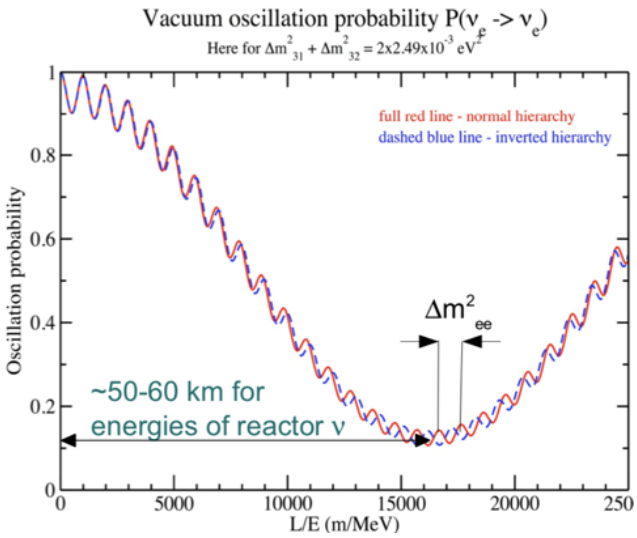

Figure 1.2 – Oscillation probability as function of $L/\mathcal{E}$ ratio for the normal or inverted ordering [6].

To achieve such a resolution, JUNO will employ a huge active volume, realized by an acrylic sphere with a diameter of 35.4 m filled with 20 kt of liquid scintillator (fig. 1.3). A larger spherical structure of ⌀19.5 m hosts 17 571 large photomultipliers (⌀20 inch) alternated by 25 600 smaller ones (⌀3 inch) for a total coverage of 78% of the total surface. The acrylic sphere and PMTs are housed in a cylindrical pool filled with ultra-pure water, that acts as a buffer to shield background radiation. Another set of 2400 large PMTs monitor the water buffer, and act as a Cherenkov detector for muon tracks, with an estimated efficiency of over 95%. On top of the detector, three plastic scintillator layers, the Top Tracker, detect cosmic muons to offer an even more reliable background removal.

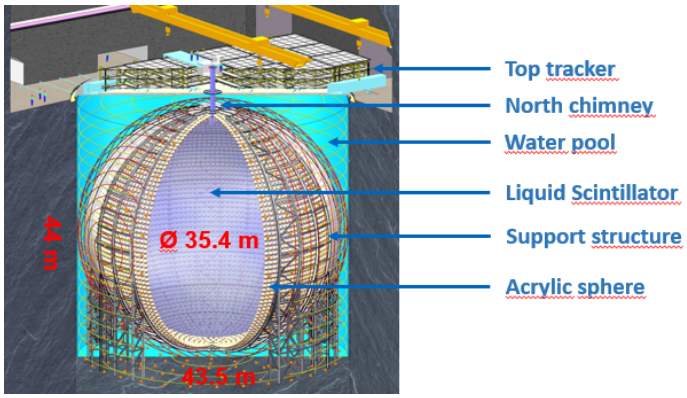

Figure 1.3 – Structure of the JUNO experiment [6].

1.2 Inverse Beta Decay

The main channel for $\overline{\nu}_e$ detection is given by the Inverse Beta Decay (IBD) interaction (fig. 1.4):

$$\overline{\nu}_e + p \rightarrow e^+ + n$$

The large active volume of JUNO is needed to have many protons available for interaction. After an IBD event, the positron promptly annihilates with an electron producing a γ rays pair, while the neutron is captured by a nucleus after an average delay of about 236 µs:

$$e^+ + e^- \rightarrow 2\gamma; \qquad n + {}^1H \rightarrow {}^2H^* \rightarrow {}^2H + \gamma(2.2\,MeV)$$

IBD has a relatively low threshold of 1.8 MeV, high cross section and is easily distinguishable from background thanks to the delayed γ signature. Almost all energy is carried by the e^+, and the total neutrino energy $\mathcal{E}_\nu$ can be reconstructed as:

$$\mathcal{E}_\nu = \mathcal{E}_{e^+} + (m_n - m_p)$$

Knowing the energies of the reactor $\overline{\nu}_e$ and the IBD cross-section, the spectrum shown in fig. 1.5 is expected.
To make discussion easier, the following notation is introduced:

- $\mathcal{E}_k$ is the kinetic energy of the emitted positron in a IBD event
- $\mathcal{E}_0 = \mathcal{E}_k + 0.511\,MeV$ is the positron total energy
- The *visible* energy in the detector is that of the positron-electron annihilation: $\mathcal{E}_{vis} = \mathcal{E}_0 + 0.511\,MeV = \mathcal{E}_k + 1.022\,MeV$

Due to the phenomenon of oscillation, and assuming a 3% energy resolution for the detector, the expected visual energy spectra in case of NO or IO are plotted in fig. 1.6.

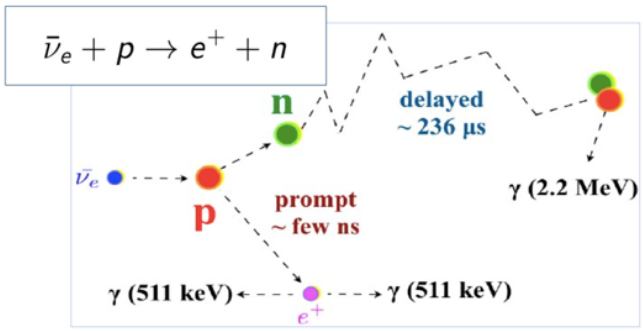

Figure 1.4 – Inverse Beta Decay interaction [6].

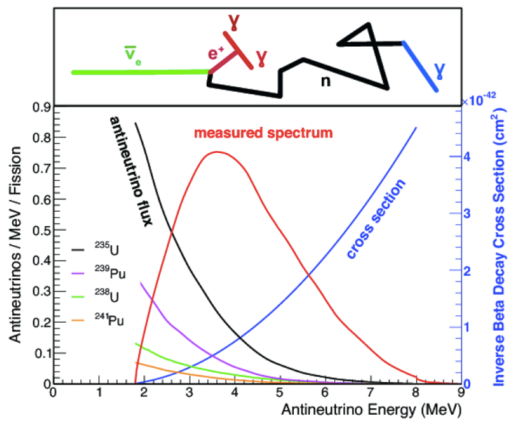

Figure 1.5 – Observed energy spectrum in absence of oscillation [6].

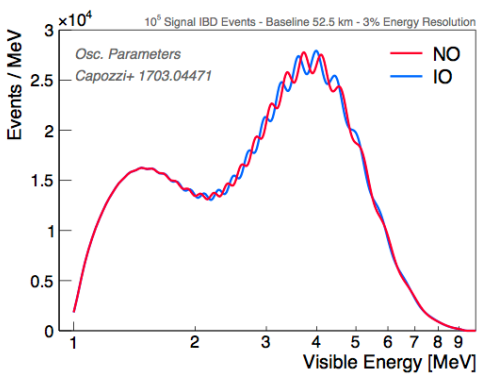

Figure 1.6 – Expected $\mathcal{E}_{vis}$ spectrum given neutrino oscillation and 3% energy resolution [7].

1.3 Scintillation detector

To estimate $\mathcal{E}_\nu$, the energy deposited in the detector is measured thanks to the *scintillation* mechanism [8], by which ionising radiation is transformed to visible photons that can be efficiently detected by PMTs. The active volume of JUNO is filled with Linear alkylbenzene (LAB), an organic light sensitive molecule that emits 280 nm radiation (in the middle UV) upon γ excitation. As organic scintillators are opaque to their light, the produced signal must be converted to a larger wavelength to be observable, and also be adapted to the PMTs input range for better efficiency. This is done using *wavelength shifter* molecules, and the fluorescence mechanism. Energy from the excited primary scintillator is transferred to secondary (and tertiary) flours through the non-radiative process of *Förster resonance energy transfer* (a kind of dipole-dipole coupling between chromophores) and then emitted as visible photons.
JUNO will employ two flours [6] (fig. 1.7): 2,5-Diphenyloxazole (PPO), with a concentration of $2.5\,\mathrm{g\,L^{-1}}$ and peak emission at 390 nm (UVA) and $3\,\mathrm{mg\,L^{-1}}$ of Bis-MSB, emitting at 430 nm (violet light).
Finally, the produced light is collected by the PMTs around the active volume, and converted to an electrical signal which is then amplified and digitized by the electronics, ready to be analyzed.
Such a setup can satisfy the optical requirements for JUNO: a light output of 10^4 photons per MeV (leading to $\sim$ 1200 PEs per MeV in the PMTs) and high transparency (attenuation length > 20 m at 430 nm).

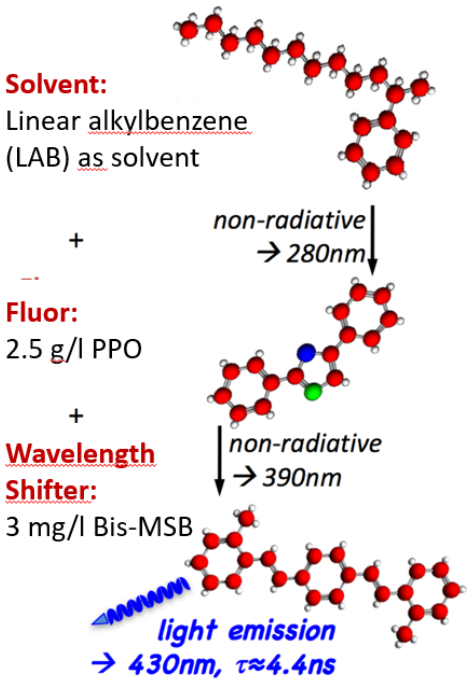

Figure 1.7 – The scintillating molecules used in JUNO [6].

Chapter 2

Frameworks

2.1 Supervised learning

The task of energy reconstruction consists of calculating $\mathcal{E}_0$ given all the available experimental information D for an event i. Several approaches exist to tackle this problem. One possibility is that of **supervised learning**, where a system is built in such a way that it automatically "learns" from given examples and extrapolates an underlying general pattern. That can be summarized in the following points:

1. **Data specification**. A set of labeled data $D_L^{\text{train}} = \{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}_{i=1,\dots,m}$ is produced. Each pair i is called an *instance*, and contains all the relevant information about the i-th event. The vector $\boldsymbol{x}^{(i)}$ represents the *features* of that event, that is a set of chosen functions of the available experimental data. On the other hand, $\boldsymbol{y}^{(i)}$ contains the values of observables of interest, such as the true energy $\mathcal{E}_0^{*,(i)}$.
   To construct D_L one needs to run an experiment with a known outcome. This can be done through a calibration procedure, or with computer simulations. The former option leads to more representative data, but it's limited in terms of control of the system's state and amount of instances generated. The latter one allows for total control of all relevant parameters and limitless instances, but no warranty to be completely representative of the real apparatus.

2. **Model specification**. A *model* $f_{\boldsymbol{W}}$ is chosen, that is a function from the vector space of features to that of labels. $f_{\boldsymbol{W}}$ is completely specified by a set $\boldsymbol{W}$ of parameters.

3. **Training**. Initially the parameters are randomly chosen from a distribution. A cost function, called *loss*, measures how far are the predictions $f_{\boldsymbol{W}}(\boldsymbol{x})$ from the known values $\boldsymbol{y}$. Then the parameters $\boldsymbol{W}$ are tweaked by an *optimizer* in order to lower the loss value. The process is iterated many times, until a good candidate for $f_{\boldsymbol{W}}$ has been found.

4. **Validation**. If the process of supervised learning succeeded, $f_{\boldsymbol{W}}$ should be able to generalize, that is predict accurate labels for features that were not part of D_L^{train}. This can be checked by evaluating the model's performance on another labeled set D_L^{val} (validation set). If the losses on both D_L^{train} and D_L^{val} are similar and too high, the model is said to have *underfit*, i.e. failed to learn enough information. However, often the performance is good on D_L^{train}, but not at all on D_L^{val}. In that case the model has *overfit*, i.e. learned useless noise patterns from data. Both cases are undesirable, and can be corrected by adding more instances to D_L^{train}, or more features per instance, or modifying the model.
   Also, the process of supervised learning requires many arbitrary choices: for example, the specific form of $f_{\boldsymbol{W}}$, the loss, the optimizer. These are called the model's *hyper-parameters*. The validation set D_L^{val} can be used to select the best options, through a *trial-and-error* process called *fine-tuning*.

5. **Testing**. The general performance of the model is evaluated one last time on a labeled set D_L^{test}, different from all the previous D_L. This is done because *hyper-parameters fine-tuning* may

specialize the model on specifics of D_L^{val} which are not representative of a general sample, leading to an overly optimistic estimate of the method's performance.

2.2 Deep Neural Networks

Deep Neural Networks are a large class of trainable pattern-recognition models that have found success in many applications in the last years, especially thanks to advances in computation hardware. The concept of an artificial neural network originated in the 60s [9], finding inspiration in the structures of the human brain.
The simplest model is called *Perceptron* [10], and consists of a single unit that computes a linear combination of its inputs $\boldsymbol{x}$ with an added offset b:

$$f(\boldsymbol{x}) = \boldsymbol{W} \cdot \boldsymbol{x} + b; \quad \boldsymbol{x}, \boldsymbol{W} \in \mathbb{R}^m, b \in \mathbb{R}$$

where the vector $\boldsymbol{W}$ contains the parameters - called *weights* - of the perceptron, and the offset b is called *bias*. Each weight w_n can be seen as the "strength" of a connection between a certain input x_n and the perceptron itself. In this picture, usually the bias is seen as the weight w_0 connecting to a constant input $x_0 = +1$ (fig. 2.1).

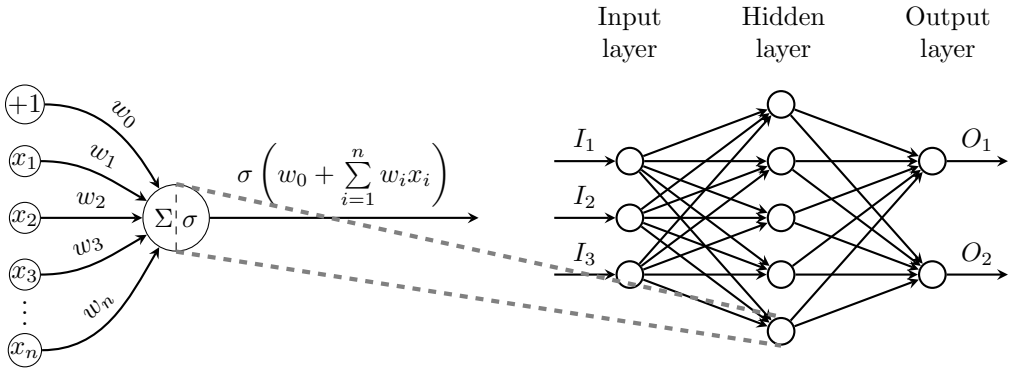

Figure 2.1 – Graph representation of a single perceptron (left) and a Multi Layer Perceptron network (right) [11].

The output is then passed to an *activation function* $h(y)$, that introduces some sort of non-linearity. The simplest possibility is given by a *step function*, that sets the output at 1 if $f(\boldsymbol{x})$ is over a certain fixed threshold τ, and to 0 otherwise. Another choice is the *sigmoid*, that is a "more gradual" step:

$$h(y) = \frac{1}{1 + e^{-y}}$$

As today, the Rectified Linear Unit (ReLU) is most used for its semplicity and effectiveness [12]:

$$\text{ReLU}(y) = \max(0, y)$$

So, the total action of the perceptron is given by the composition of the linear unit and the activation:

$$P(\boldsymbol{x}) = (h \circ f)(\boldsymbol{x})$$

Many perceptrons can be connected to form a *Multi Layer Perceptron* (MLP). Units are organized in *layers*, such that each perceptron receives all the outputs from the previous layer. The first and last layer make up respectively the input and output, and any layer in between (if present) is an *hidden* layer.
Let $N_j + 1$ be the number of units of the j-th layer, with the 0-th unit always outputting a constant 1 (for the bias). As biases only act as inputs for the following layers and do not receive connections from the previous ones, the output of the j-th layer will be a vector $\boldsymbol{a}^{(j)} \in \mathbb{R}^{N_j}$.

Let then $w_{ik}^{(j)} \in \mathbb{R}$ be the weight connecting the k-th input of the j-th layer to the i-th unit of the following $(j+1)$-th layer. For a given layer j, the set of $w_{ik}^{(j)}$ with $0 \leq k \leq N_j$ and $1 \leq i \leq N_{j+1}$ forms a $N_{j+1} \times (N_j + 1)$ matrix $\boldsymbol{W}^{(j)}$ called the weights matrix. Knowing $\boldsymbol{a}^{(j)}$, it is now possible to compute $\boldsymbol{a}^{(j+1)}$ as:

$$\boldsymbol{a}^{(j+1)} = h\left[W^{(j)}\begin{pmatrix}1\\ \boldsymbol{a}^{(j)}\end{pmatrix}\right]$$

where h is the activation function, applied element-wise.
MLPs of sufficient size can model arbitrary functions, given a proper choice for the weights. These can be learned through supervised learning if a sufficiently sized labeled set D_L is available. Weights are initialized randomly, and the *loss* function $\mathcal{L}$ is computed over the entire D_L^{train}. Then one proceeds to calculate the derivatives $\partial\mathcal{L}/\partial w_{ik}^{(j)}$ (for example using the *backpropagation* algorithm, or numerical methods such as *autodiff*), and weights are slightly nudged to reduce $\mathcal{L}$. That is the gist of the *gradient descent* algorithm.
To achieve faster convergence, an optimizing step may be made using subsets of D_L^{train} with size s (*batch size*), leading to the *stochastic gradient descent algorithm* [13].

MLPs are an example of *Full Connected Networks*, as all units are connected to every other in the adjacent layers. This gives huge flexibility to the model, but also increases computational cost and can lead to *overfitting*. If input data is of very high dimensionality, such as in the case of high resolution images, MLPs prove difficult to train, and need huge datasets to reach useful results. Also, MLPs cannot exploit symmetries, and are sensitive to small changes in the inputs.
All these problems can be alleviated by using instead a *Convolutional Neural Network* (CNN) [14]. Inspired by neurons in the visual cortex, CNNs make use of *partial connections* and *weight sharing* to reduce network complexity.
Input data in a CNN is usually a multichannel image, that is a 3D tensor $\boldsymbol{X} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ [13], where d_1 and d_2 are the image's dimension, and d_3 is the number of channels.
Each unit in a *convolutional layer* is connected only to a small square window of input pixels, the *receptive field* or *kernel*, and weights are shared between all units. This limits the number of free parameters, and also leads to some kind of translational invariance, as equal patterns in different positions produce the same activation of different neurons.
This is mathematically equivalent to computing the cross-correlation of $\boldsymbol{X}$ with a small 3D tensor $\boldsymbol{F}_k \in \mathbb{R}^{w \times w \times d_3}$ (*filter*), where w is the kernel size, leading to an output $\boldsymbol{O}^k \in \mathbb{R}^{(d_1-w+1)\times(d_2-w+1)}$:

$$O_{ij}^k = \langle [\boldsymbol{X}]_{ij}, \boldsymbol{F}_k \rangle = \sum_{i'=1}^{w}\sum_{j'=1}^{w}\sum_{l=1}^{d_3} [\boldsymbol{X}]_{i+i'-1,j+j'-1,l}[\boldsymbol{F}_k]_{i',j',l}$$

where $[\boldsymbol{X}]_{ij} \in \mathbb{R}^{w \times w \times d_3}$ is a small square "patch" of $\boldsymbol{X}$ starting at pixel (i, j), and $\langle \cdot, \cdot \rangle$ is the inner tensor product (tensor contraction).
Usually, convolutional layers make use of a number $K \geq 1$ of filters, and all their outputs are simply stacked on different channels, leading to $\boldsymbol{O} \in \mathbb{R}^{(d_1-w+1)\times(d_2-w+1)\times K}$, which is then transformed by an element-wise *activation function*.
To detect patterns at different sizes, and to exploit symmetries, CNNs employ *Pooling layers*, that "distill" the outputs of convolutional layers. This is done by sliding a window over the input and computing a statistic (such as maximum or average) over the selected values. For example, using a 2×2 kernel size leads to an output of half the resolution, that hopefully preserves the main characteristics of interest.
More details on these architectures, and on modern training techniques are referred to literature [15].

2.2.1 Technical details

Several efficient libraries to create, train and evaluate Deep Neural Networks are already available for many programming languages. In this work, Tensorflow for Python 3.6.7 [16] is used, together with its optimized implementation of the Keras API.
Training is done on a virtual machine hosted on Cloud Veneto, using a Nvidia Titan Xp GPU with 12 GB of vRAM, a 8 cores CPU, 40 GB of RAM and 500 GB of available storage on a SSD. In fact, as Deep Neural Networks involve lots of matrix operations, a GPU can provide a significant speed up for calculations - achieving up to a 10 fold gain on performance compared to a normal CPU.

Chapter 3

Analysis

3.1 Datasets

All the labeled datasets used in the following analysis are generated by means of Monte Carlo simulation through the SNiPER software [17]. Specifically:

- D_L^{train} and D_L^{val} are formed by partitioning a set $\boldsymbol{S_A}$ of 10^6 instances with uniformly distributed energies $\mathcal{E}_k^* \in [0, 10]$MeV.
- D_L^{test} consists of 10 sets $\boldsymbol{S_B}$, each with $2{\cdot}10^3$ instances each and discrete energy $\mathcal{E}_k^* \in \{0, 1, 2, \dots, 9\}$MeV.

To simplify analysis, the following points are made:

- Only **large PMT**s (⌀20 inch) are considered, while ignoring the rest (⌀3 inch). That reduces the size of datasets, and prevents a more difficult analysis of two different classes of features.
- Only events with vertices inside a **fiducial volume** (FV, defined as $R^* < 17.2\,\text{m}$) are considered, at the cost of losing $\sim 8\%$ of data. That is done because photons emitted near the edge of the acrylic sphere tend to be reflected at the interface and absorbed by the liquid, vastly reducing the amount of detected photoelectrons and the accuracy of energy reconstruction.
  Fortunately, previous results [18][19] show that events inside/outside the FV can be reliably distinguished by experimental data alone, through the use of an appropriate classifier. Hence, this *a priori* cut can be made even with real data.
- PMTs are sometimes triggered even in absence of any signal, because of leakage currents, thermal noise or other effects [20]. Specifically, of the 18k large PMTs, 25% of them are made by Hamamatsu, and the remaining 75% by NNVT, respectively with measured dark rates of $\sim 16\,\text{kHz}$ and of $\sim 50\,\text{kHz}$. Such *Dark Noise* (DN) is simulated in the available datasets, but it is totally removed for the first analysis, and will be examined later in section 3.4.

3.2 Baseline model

A first approach for energy reconstruction is based upon [19], and it is replicated in this section. The obtained results will then be used as baseline for comparison with a different model.
For this method, a "minimalistic" set of **features** for each event is selected, consisting in:

1. Number of photo-electrons detected (`totalPE_lpmt`)
2. Average hit time (`ht_mean`) in ns, relative to the first hit time of the first fired PMT.
3. Center-of-hits radius (`cohR`) and z-coordinate (`cohZ`).
   Let N_{PMT} be the number of PMTs, $\{\boldsymbol{r_j}\}_{j=1,\dots,N_{\text{PMT}}}$ their positions (in mm) relative to the center

of the detector, and $\{n_j\}_{j=1,\dots,N_{\text{PMT}}}$ the number of hits for each PMT. Then, the center-of-hits is defined as the vector:

$$\mathbf{CoH} = \frac{1}{N_{\text{PE}}} \sum_{j=1}^{N_{\text{PMT}}} \boldsymbol{r_j} n_i \qquad N_{\text{PE}} = \sum_{j=1}^{N_{\text{PMT}}} n_j$$

`cohR` is defined as the norm of **CoH**:

$$\text{cohR} = \|\boldsymbol{CoH}\|$$

All features are normalized to have 0 average and unit standard deviation:

$$\boldsymbol{x}' = \frac{\boldsymbol{x} - \bar{\boldsymbol{x}}}{\boldsymbol{\sigma_x}}; \qquad \bar{\boldsymbol{x}} = \frac{1}{m}\sum_{j=1}^{m} \boldsymbol{x}^{(j)}; \quad \boldsymbol{\sigma_x} = \frac{1}{m-1}\left(\sum_{j=1}^{m} (\boldsymbol{x}^{(j)} - \bar{\boldsymbol{x}}) \odot (\boldsymbol{x}^{(j)} - \bar{\boldsymbol{x}})\right)^{\frac{1}{2}} \tag{3.1}$$

where square root and multiplication ($\odot$) are computed element-wise.
The only **label** is given by the total positron energy $\mathcal{E}_0^*$, without any normalization.

D_L^{train} and D_L^{val} are generated by a $75\% - 25\%$ split of S_A after the FV cut and normalization, resulting in $\sim$ 688k instances for training and $\sim 229k$ for validation.

The model consists of a Fully Connected Neural Network with 5 hidden layers of 128 neurons each, using the ReLU activation function, and is implemented in Keras 2.2.4-tf within Tensorflow 2.0.0-beta1 [16]. The loss function is the Mean Absolute Percentage Error (MAPE):

$$\text{MAPE} = \frac{1}{m}\sum_{i=1}^{m} \left| \frac{f(\boldsymbol{x}^{(i)}) - y^{(i)}}{y^{(i)}} \right|$$

where m is the number of instances in D_L. MAPE is chosen because it is not sensitive to outliers, as would be the Mean Squared Error (MSE). Also, MAPE minimizes the relative error rather than the absolute one (computed in the Mean Absolute Error - MAE) leading to a better performance in the task of energy resolution. However, MAPE makes convergence more difficult than the alternatives: depending on the weights initialization, it can be necessary to train the model with MAE in the first few epochs, and then switch to MAPE in the proximity of the global minimum.
The optimizer is Nadam (Adam with Nesterov acceleration) [21], with initial learning rate of 10^{-3}, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate is halved every 5 epochs where the validation loss does not decrease.
Convergence is reached after 50 epochs with batch size of 64, reaching a min validation loss of ~ 1.37.

The model's performance is evaluated on D_L^{test}, consisting of instances of S_B after the FV cut. Test features are normalized using the previously computed average and standard deviation for training and validation by applying (3.1).
For each $\mathcal{E}_k^*$:

1. Predictions $\{y_i\}_{i=1,\dots,m}$ for $\mathcal{E}_0$ are computed using the model and plotted as an histogram.
2. Let $\bar{y}$ be the average of $\{y_i\}$ and σ_y their standard deviation. A gaussian is fitted to predictions in $[\bar{y} - 4\sigma_y, \bar{y} + 4\sigma_y]$, leading to estimates for the centroid $\mu \pm \sigma_\mu$ and the width $\sigma \pm \sigma_\sigma$ (fig. 3.2). The presence of outliers will be analyzed later.
3. The energy resolution at $\mathcal{E}_{\text{vis}}^* = \mathcal{E}_k^* + 1.022\,\text{MeV}$ is computed as:

$$\sigma_{\%} = \frac{\sigma}{\mathcal{E}_{\text{vis}}^*} \cdot 100$$

where σ is the estimate computed at point 2. A plot of $\sigma_{\%}(\mathcal{E}^*_{\text{vis}})$ together with the sample standard deviation σ_y is shown in fig. 3.1. The red dashed line is the expected energy resolution given by the detector's characteristics, as reported in [1, p. 195]:

$$\frac{\sigma}{\sqrt{\mathcal{E}_{\text{vis}}}} = \sqrt{\left(\frac{2.68}{\sqrt{\mathcal{E}_{\text{vis}}}}\right)^2 + \left(\frac{0.9}{\mathcal{E}_{\text{vis}}}\right)^2} \tag{3.2}$$

4. Mean energy bias is defined as the normalized distance of the mean predicted visual energy from the true value $\mathcal{E}^*_{\text{vis}}$. As the predicted $\mathcal{E}_0$ differs from $\mathcal{E}_{\text{vis}}$ by a constant ($\mathcal{E}_{\text{vis}} = \mathcal{E}_0 + 0.511\,\text{MeV}$), the following formula can be used:

$$\text{Bias}_{\%} = \frac{\mu - \mathcal{E}^*_0}{\mathcal{E}^*_{\text{vis}}}$$

where μ is the estimate computed at point 2. A plot of the energy bias is shown in fig. 3.3.

Several observations can be made thanks to fig. 3.1, 3.2, and 3.3:

- Predicted energies $\mathcal{E}_0$ for the first test dataset ($\mathcal{E}^*_k = 0\,\text{MeV}$) are not normally distributed around $\mathcal{E}^*_0 = 0.511\,\text{MeV}$ (fig 3.2). That happens because values $\mathcal{E}_0 < \mathcal{E}^*_0$ are physically impossible. The gaussian fit is then not representative, and a better estimate for energy resolution is given by using the sample standard deviation (σ_y) instead of the fitted σ.

- For most points, σ_y and σ are close (fig. 3.1), but for $\mathcal{E}_{\text{vis}} \in \{4, 7, 10\}\text{MeV}$ σ_y is significantly larger than σ. That is due to the presence of **outliers**: predicted values that are outside a 4σ region centered on the mean. Each sample comprises ~ 1800 instances, and so these points should be very unlikely (frequency outside 4σ range is $1 : 15\,787$). However, for $\mathcal{E}_{\text{vis}} = 7\,\text{MeV}$ there are 2 outliers of this type.
  All these outliers have predicted energies much lower than the sample mean. Table 3.1 shows their relevant features. It can be noted that all of them happen at a radius near the FV cut, and are cases where a significant part of the event's energy is not detected, making it impossible to reach a satisfying reconstruction.
  The problem of distinguishing outliers from experimental data alone is not of easy resolution, and will not be addressed in this work.

- Energy biases are normally distributed around 0 (fig. 3.3), with deviations not higher than 0.1%. This is well within the JUNO expected energy accuracy of 1%.

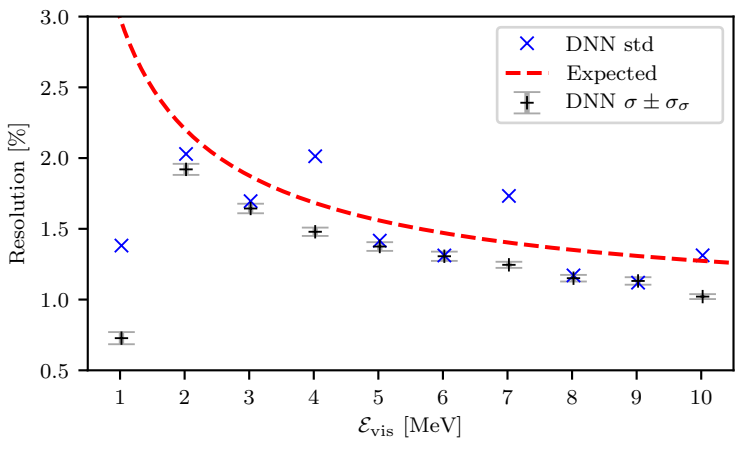

Figure 3.1 – Energy resolution for the Baseline model

$\mathcal{E}^*_{\text{vis}}$ [MeV]	$\mathcal{E}_0$ pred. [MeV]	$\frac{N_{\text{PE}} - \bar{N}}{\sigma}$	R^* [m]
4	1.03	−9.01	16.81
7	5.88	−2.71	16.58
7	2.97	−7.30	17.19
10	7.07	−5.19	16.79

Table 3.1 – Relevant data for all the observed outliers for the baseline model: true visual energy ($\mathcal{E}^*_{\text{vis}}$), predicted $\mathcal{E}_0$, number N_{PE} of collected photons and true radius R^*. For each event, $\bar{N}$ is the average number of collected PE at the same energy is computed, and σ is the corresponding standard deviation.

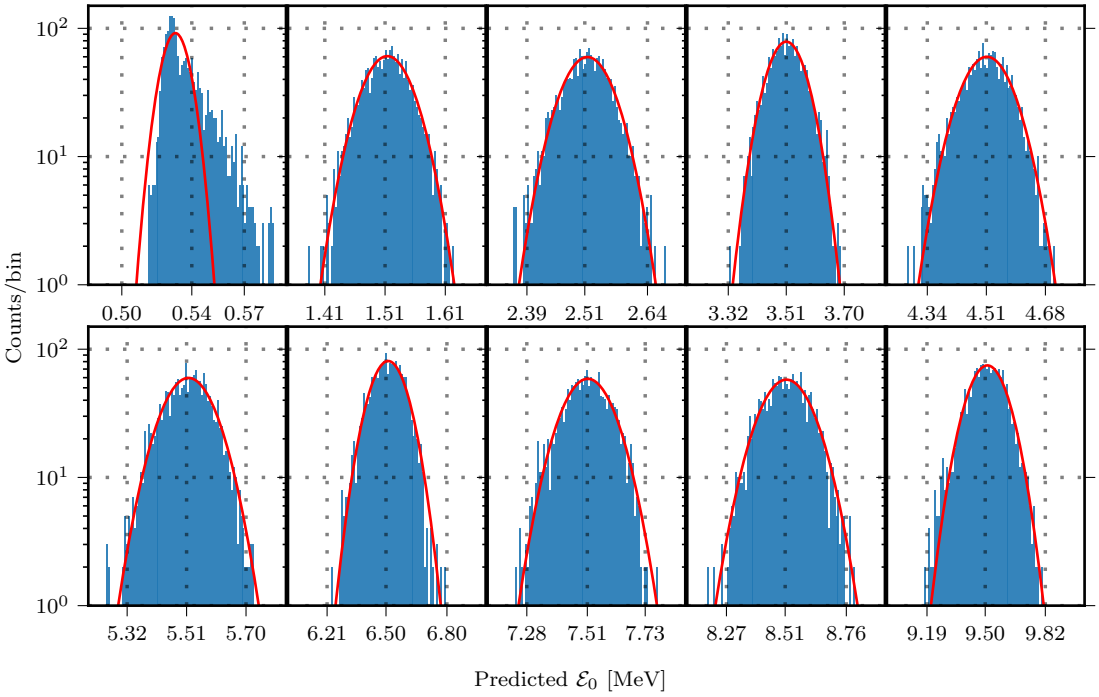

Figure 3.2 – Gaussian fit for the predictions of the Baseline model for D_L^{test}

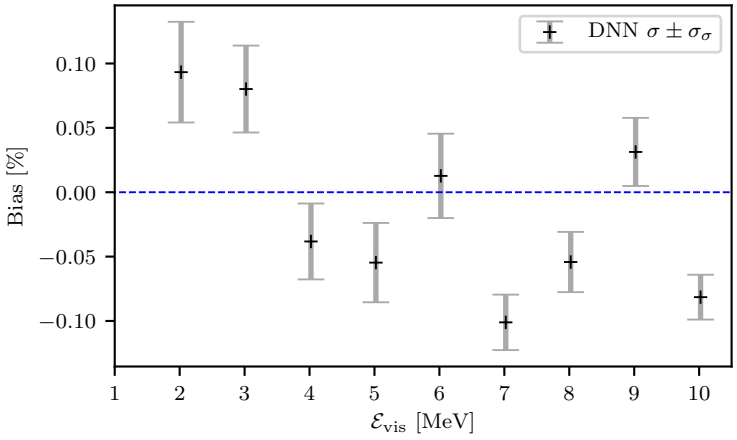

Figure 3.3 – Energy bias for the baseline model

3.3 Spherical model

Deep Neural Networks are suited for complex analysis of a large number of features, and are able to extract meaningful representations from raw data. In the following section, a "maximal" information approach is attempted for energy reconstruction.

3.3.1 Data preparation

The individual response of each PMT can be simulated using Monte Carlo methods. For each event, every photoelectron collected by the detector is tagged by the PMT hit (`pmt_id`), hit time (`hit_time`, in ns, relative to the instant of positron emission) and origin, whether signal or noise (`isDN`). PMTs positions relative to the detector center are known, and so signals can be reconstructed on a spherical surface.
It is not trivial how to use this kind of data with a DNN, and a reasonable representation is needed. The two main challenges are:

1. **Spherical signal**. A valid model should be able to learn spatial correlations and exploit rotational invariance. In fact, the predicted energy should not change after rotating the signal.
2. **Time dependence**. Temporal correlations should be exploited to improve energy reconstruction and better distinguish signal from noise.

Convolutional Neural Networks have achieved state of the art results for many pattern recognition tasks, given their ability to harness spatial and temporal correlations. However, CNNs are usually employed on Euclidean geometries, where they exhibit translational invariance. Rotations are considerably more difficult, as no completely symmetrical discretization of the sphere can be made. For example, while it is easy to compute a "translation by one pixel", an analogue "rotation by one pixel" is ill-defined - therefore a naive approach for computing spherical cross-correlation is impossible. Notwithstanding, it is possible to define spherical convolutions by multiplication in the spectral domain. That is usually a computationally intensive task, even using Fast Fourier Transforms. Fortunately, an alternative and quicker approach to reach rotational invariance exists, as demonstrated in the DeepSphere model [22].

Deepsphere is a Spherical Convolutional Neural Network optimized for the HEALPix discretization, which stands for a Hierarchical, Equal Area and iso-Latitude Pixelization of the sphere, developed by NASA for astrophysics research [23]. The HEALPix scheme starts by dividing the spherical surface in 12 equal area quadrilaterals of varying shape, organized in three rings - two around the poles and one around the equator (fig. 3.4). Each pixel can then be divided in 4 equal area parts, and the process repeated until the desired resolution is achieved. The number of divisions along the side of one of the original 12 pixels is the N_{side} parameter, which needs to be a power of 2 to preserve the hierarchical structure. The total number of partitions is then $N_{\text{pix}} = 12N_{\text{side}}^2$.

Given the available simulated data, conversion to the HEALPix discretization is achieved using methods from the Healpy library [24] in Python 3.6.
For the following model $N_{\text{side}} = 16$ is chosen, corresponding to $N_{\text{pix}} = 3072$, or 5.77 PMTs per pixel. That allows to retain most of the information about individual PMTs, while producing datasets of a more manageable size. In fact, to have at most one PMT per pixel one would need to choose $N_{\text{side}} = 64$, leading to a whopping $N_{\text{pix}} = 49\,152$.

The following features are selected:

- **Total charge**. For each spherical pixel the total charge (number of PE hits) of the corresponding PMTs is computed. All hits due to Dark Noise are ignored.
- **First hit time**. Each spherical pixel is associated to the minimum first hit time of the corresponding PMTs, rounded to the nearest ns, where 0 ns is the instant of positron emission

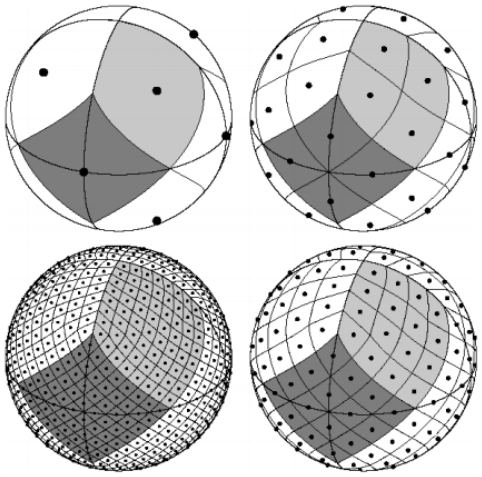

Figure 3.4 – From top moving clockwise: partitions corresponding to $N_{\text{side}} = 1, 2, 4, 8$. All pixels have equal area, and their centers (the black dots) are organized in iso-latitude rings. One of the polar base-resolution pixel is colored in light-gray, while an equatorial one is in dark-gray. To increment resolution by one step, every pixel is split in 4 parts. (Image taken from [23]).

(assumed known). That is done because usually the most relevant information is given by the first hit, as subsequent ones can be reflections and/or noise.
However, experimentally the exact instant for the IBD cannot be measured, and so hit times must be expressed relative to an arbitrary time. That adds noise to the dataset, and its effect on performance will be studied later in sec. 3.4.
Pixels that correspond to PMTs that are never hit are assigned (arbitrarily) a first hit time value of 1024 ns. This is done for two reasons: such a value is easily distinguished from all other data, and it is large enough to form a visual gradient, as all events result in a pixel hit first, surrounded by pixels hit gradually at later times. Hence, in a certain sense, empty pixels are hit "at infinity".

Charge $\boldsymbol{c}$ and hit times $\boldsymbol{h}$ vectors, with dimension $= N_{\text{pix}}$, are collected in two spherical channels for each event: $\boldsymbol{x}^{(i)} = (\boldsymbol{c}^{(i)}, \boldsymbol{h}^{(i)})$ (fig. 3.5).

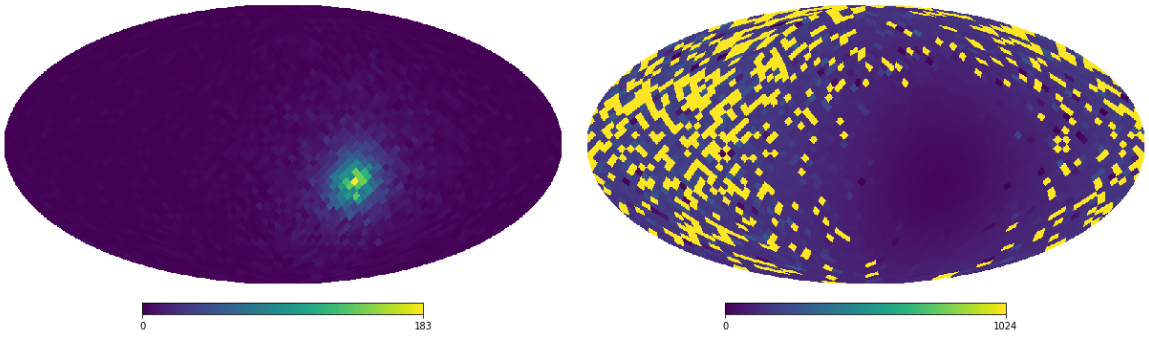

Figure 3.5 – Mollweide Projection of spherical channels: total charge per pixel (left) and first hit time (right)

The only **label** is, as before, $\mathcal{E}_0^*$.

A dataset S_C is produced using 200k events from S_A and applying the FV cut, leading to a total of $\sim$ 180k instances. Features are normalized by scaling each channel maximum to 1:

$$\boldsymbol{x}' = \left(\frac{\boldsymbol{c}}{c_{\text{max}}}, \frac{\boldsymbol{h}}{h_{\text{max}}} \right) \tag{3.3}$$

where c_{max} and h_{max} are the maximum values of charge/first hit amongst all instances and pixels of the entire dataset.
S_C is then split in 160k and 38k labeled datasets for training and validation.

3.3.2 Model architecture and hyper-parameters

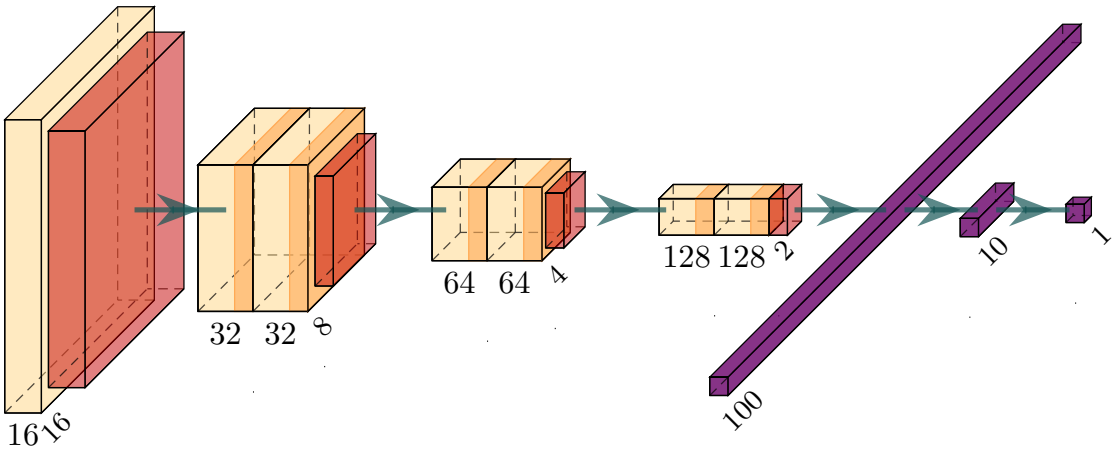

Figure 3.6 – Representation of the spherical model's architecture. *Convolutional* layers are in yellow, *maxpool* in red and *fully connected* in purple. Number of filters is reported below each convolutional layer, as their size in N_{side} units (the sloped number). For fully connected layers, the sloped number represents the number of neurons. Plotted using [25].

The Deepsphere model is created using the publicly available code in [26] and Tensorflow 1.14 [16], following the published example for multi-channel regression.
The architecture is made of a convolutional block, followed by a fully connected segment (fig. 3.6):

1. The **CNN Block** contains 7 layers with ReLU activation, the first with 16 filters, followed by couples of layers with doubled filters, so that the last two layers have 128 filters. Before every doubling a Maxpooling layer halves the number of pixels. Also, batch normalization is executed after each layer.

2. The **fully connected** segment is made of three layers with ReLU activation, with 100, 10 and 1 neurons respectively. The last layer has no activation, and it is used as the regression output.

Between the CNN block and the full connected segment there is a single *statistical layer*, as specified by the DeepSphere model, that averages the previous filters to preserve rotational invariance.
The model's architecture is inspired from that of LeNet [13], with the number of filters doubled after every maxpooling layer to maintain a constant complexity. Many different alternatives, especially in full-connected segment, are evaluated on D_L^{val}, and the one here presented is the best performing found. In fact, adding more dense layers at the end generally leads to overfitting, without significant improvement even on the training loss.

For the loss function Mean Absolute Error (MAE) is used, that is the normalized ℓ_1 distance between predictions and labels:

$$\text{MAE} = \frac{1}{m}\sum_{i=1}^{m} |f(\boldsymbol{x}^{(i)}) - y^{(i)}|$$

MAE is here preferred over MAPE because it converges faster and more reliably while maintaining a similar performance.

Training is done with a batch size of 64 (which is both a factor of D_L^{train} size and a power of 2, for better speed and stability), and the Adam optimizer, with initial learning rate of 0.1 and $\beta_1 = 0.8$, $\beta_2 = 0.9$, $\epsilon = 10^{-3}$. Learning rate is decayed by a factor of 0.95 at the end of every epoch, and convergence is reached after ~ 60 epochs. Training takes about 8 hours on a Titan Xp GPU.

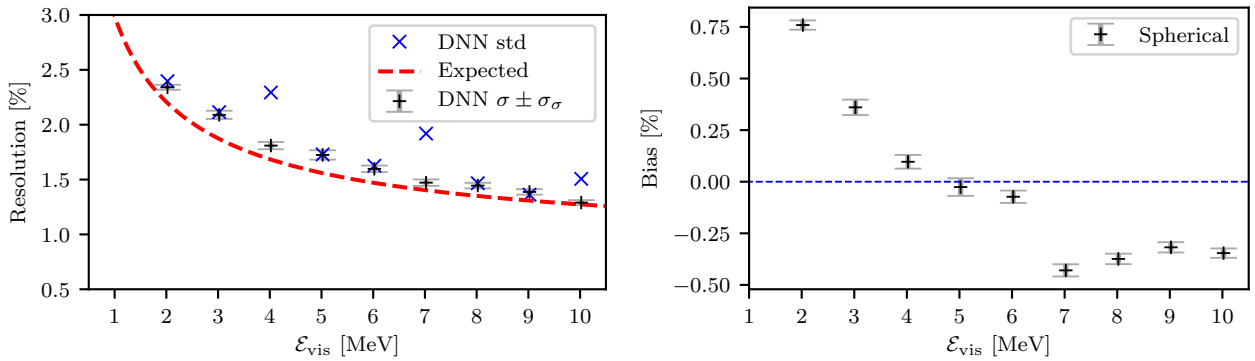

Figure 3.7 – Energy resolution (left) and bias (right) for the spherical model, in the absence of Dark Noise.

3.3.3 Computing Model Performance

The same dataset D_L^{test} introduced for the baseline model is used to compute the features needed for the spherical model, which are normalized using the same parameters h_{max} and c_{max} calculated during training (3.3). Evaluation proceeds with the same method as before, leading to a plot of energy resolution and energy bias (fig. 3.7). The point at $\mathcal{E}_{\text{vis}} = 1\,\text{MeV}$ lies at the edge of the training dataset, leading to an estimate which is not representative of the reconstructed resolution, and is therefore omitted.
As before, for $\mathcal{E}_{\text{vis}} \in \{4, 7, 10\}\text{MeV}$ the sample standard deviation of predictions differs significantly from the σ of the fitted gaussian. This is due to the same 4 outliers already discussed for the baseline model.
While the resolution performance is comparable to that of the baseline model, the energy bias for the spherical architecture is much higher. This could be due a slight overfit, or an unfortunate weights initialization. However, for the mid energies ($4 - 6\,\text{MeV}$) the bias is well within the 1% required range.

3.4 Response to Dark Noise

The presence of Dark Noise in PMTs introduces perturbations in the training data, making the task of energy reconstruction significantly harder.
Following the same procedure of the previous section, both Baseline and Spherical model are trained and tested on datasets that incorporate Dark Noise.
Regarding the spherical model, an additional preprocessing step is applied to data:

- All hit times are expressed relative to the start of a peak in the detection rate. More precisely, the simulated hit times for a certain event (with 0 ns corresponding to the instant of positron emission) are counted in bins b_i of 1 ns width, each starting from $(t_{\min} + i)$ns with $t_{\min} \equiv -100$ ns. Let $b_{i'}$ be the first bin containing more hits than an arbitrary threshold of 15. Then $(t_{\min} + i')$ns is set as the new time reference *zero* for that event. That procedure limits the information available to the model to that which is experimentally measurable.
  The threshold can be any number larger than the dark noise hit rate per ns, but lower than the maximum hit rate per ns of any event in the dataset. The chosen value of 15 satisfies both requisites, and is found by manually examining the hit times histogram of several events.
  Using the new hit times, and following the suggestion in [27], all hits after 300 ns are ignored, and do not contribute to total charge nor first hit time. Most of late hits, in fact, are due to Dark Noise, as can be seen in fig. 3.8. This kind of data rejection can be easily implemented on experimental data, leading to a slightly clearer signal, especially in the areas that receive few hits.

Both models are trained using the same hyper-parameters as before. The added disturbance make the spherical model's training longer, now requiring ~ 100 epochs to reach convergence.
The results of evaluation are plotted in fig. 3.9 and 3.10. The following observations can be made:

- Both models perform well in the presence of Dark Noise, reaching comparable resolutions. The baseline model suffers most from the added perturbation, but still achieves a slightly better result than the spherical one.
  This could be due the following:
  - The spherical model is trained on only 160k instances, versus the 750k of the baseline model.
  - The resolution given by $N_{\text{side}} = 16$ is not sufficient to represent data from single PMTs, leading to a loss of useful information.

  These limitations were imposed to simplify the model's training. In fact, the Deepsphere code can be directly used only if the entire dataset is loaded in RAM, limiting the number of instances and their resolution. Also, training time needs to be considered, as hyperparamethers fine-tuning, which was necessary to find the best architecture, requires the evaluation of many different models.
  Nonetheless, the spherical model proves to be efficient at generalization: even with much more complexity and a smaller dataset it manages to only slightly overfit, reaching a training MAE loss of 0.065 versus 0.084 for the validation set. This is thanks to rotational invariance, and the relative low number of units in the full-connected segment.

- Bias ranges are similar and well under the 1% threshold for both methods. The baseline model systematically underestimates the energy: this can be due the added variance from dark noise, and the fact that the MAPE loss function tends to penalize overestimates more than underestimates. Also the spherical model tends to predict lower energies, but this could be due to a slight overfit of events at low $\mathcal{E}_{\text{vis}}$, that could be corrected by using a larger dataset for training.

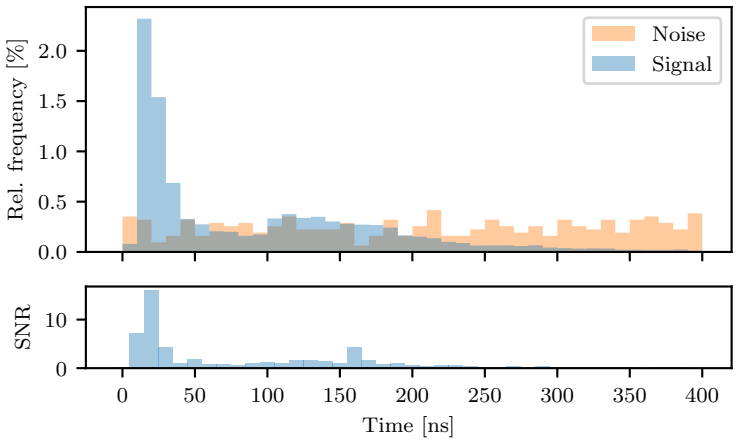

Figure 3.8 – Histograms for signal and DN hits for a certain event. Normalization is applied for better visualization, and alters the Signal to Noise Ratio (SNR) by a constant factor. As observed, after 300 ns most hits are due to DN.

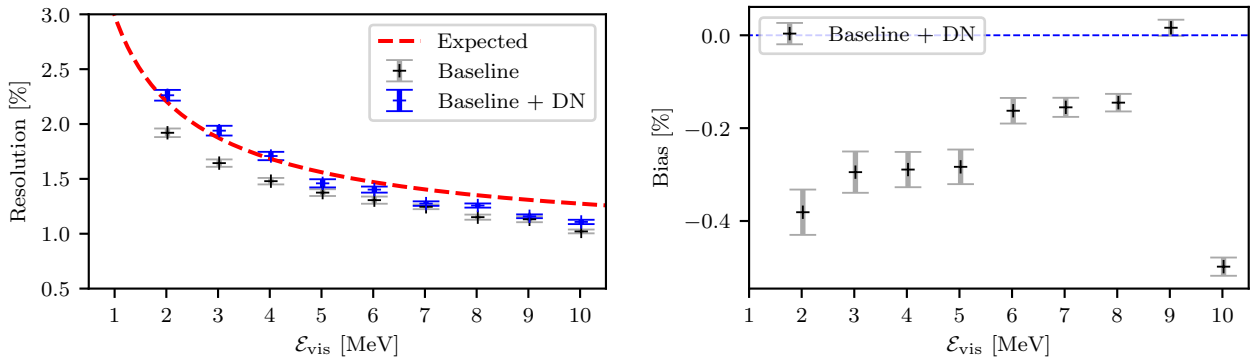

Figure 3.9 – Energy resolution (left) and bias (right) for the Baseline model in the presence of DN. Resolution without DN is reported as comparison.

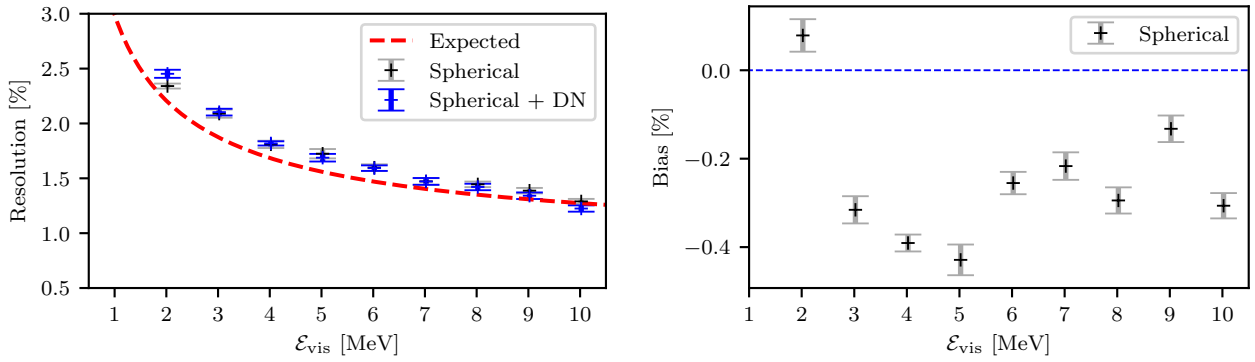

Figure 3.10 – Energy resolution (left) and bias (right) for the Spherical model in the presence of DN. Resolution without DN is reported as comparison.

Chapter 4

Conclusions and Outlook

Two approaches using Deep Neural Networks have been evaluated on simulated IBD events. The first is a *baseline* model, based upon previous work, that consists of a Fully Connected Neural Network trained on 750k instances and using only a "minimalistic" set of features - such as the total collected charge in an event, and the mean hit time of all PMTs. This method proves to be sensitive to dark noise, but still achieves a visual energy resolution of $(2.26 \pm 0.05)\%$ at 2 MeV, comparable to the 2.2% expected from theory (3.2).
The second approach employs a Convolutional Neural Network with spherical invariance, based upon the Deepsphere model, which uses a "maximal" set of features - that is position and timing data from each PMT. This model proves to be insensitive to dark noise, and achieves a good resolution of $(2.45 \pm 0.03)\%$ at 2 MeV with a much smaller dataset of only 160k instances. Hence, rotational invariance should be a desired property for a Deep Neural Network. However, the more complex architecture of such a spherical model makes it more difficult and computationally costly to train.
In the end, the baseline model achieves the best results for energy reconstruction - proving that there could still be some margin of improvement for more complex methods. Specifically, the following points can be expanded in future work:

- Using more instances to train the spherical model, and a better resolution - associating each spherical pixel with at most one PMT - at the cost of much larger datasets and longer training times. This will likely require adapting the Deepsphere internal code, as currently it can be easily used only if all data is loaded in RAM.
- Using a custom graph for the PMTs positions, without having to resort to the HEALPix discretization.
- Using *dropout layers* to generate *ensembles* for the network. That would allow to estimate the confidence for reconstructed energies, predicting results with error-bars.

Bibliography

[1] F. An et. al The JUNO collaboration. *Neutrino Physics with JUNO*. J. Phys. G 43 (2016) 030401. 2015. DOI: 10.1088/0954-3899/43/3/030401. eprint: arXiv:1507.05613.

[2] The Super-Kamiokande Collaboration and Y. Fukuda et al. "Evidence for oscillation of atmospheric neutrinos". *Phys. Rev. Lett.* 81 (1998), p. 1562. DOI: 10.1103/PhysRevLett.81.1562. eprint: arXiv:hep-ex/9807003.

[3] Z. Maki, M. Nakagawa, and S. Sakata. "Remarks on the unified model of elementary particles". *Prog. Theor. Phys.* 28 (1962). [,34(1962)], pp. 870–880. DOI: 10.1143/PTP.28.870.

[4] S. M. Bilenky. "Bruno Pontecorvo and the neutrino". *Usp. Fiz. Nauk* 184.5 (2014), pp. 531–538. DOI: 10.3367/UFNe.0184.201405g.0531.

[5] B. Pontecorvo. "Neutrino Experiments and the Problem of Conservation of Leptonic Charge". *Zh. Exsp. Teor. Fiz.* 53 (1967), pp. 1717–1725.

[6] Hans Th. J. Steiger. *Prospects, Design and Status of JUNO. 8th International Conference on New Frontiers in Physics (ICNFP 2019)*. Aug. 27, 2019. URL: https://indico.cern.ch/event/754973/contributions/3541041/attachments/1897893/3131760/go.

[7] Wenjie W. *Neutrino Oscillation Studies in JUNO. NuFact 2019, Daegu, South Korea*. Aug. 27, 2019. URL: https://indico.cern.ch/event/773605/contributions/3477836/attachments/1897539/3131058/JUNO-NuOscPhysics.pdf.

[8] C. Grupen and B. Shwartz. *Particle Detectors*. 2nd ed. Cambridge Monographs on Particle Physics, Nuclear Physics and Cosmology. Cambridge University Press, 2008. DOI: 10.1017/CBO9780511534966.

[9] S. Juergen. "Deep Learning in Neural Networks: An Overview" (2014). DOI: 10.1016/j.neunet.2014.09.003. eprint: arXiv:1404.7828.

[10] C. A. L. Bailer-Jones, R. Gupta, and H. P. Singh. *An introduction to artificial neural networks*. 2001. eprint: arXiv:astro-ph/0102224.

[11] PetarV-. *Multilayer Perceptron (MLP)*. https://github.com/PetarV-/TikZ/tree/master/Multilayer%20perceptron. 2016.

[12] X. Bing et al. *Empirical Evaluation of Rectified Activations in Convolutional Network*. 2015. eprint: arXiv:1505.00853.

[13] F. Jianqing, M. Cong, and Z. Yiqiao. *A Selective Overview of Deep Learning*. 2019. eprint: arXiv:1904.05526.

[14] Y. LeCun et al. "Handwritten Digit Recognition with a Back-Propagation Network". *Advances in Neural Information Processing Systems 2*. Ed. by D. S. Touretzky. Morgan-Kaufmann, 1990, pp. 396–404. URL: http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network.pdf.

[15] G. Ian, B. Yoshua, and C. Aaron. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[16] A. et al. Martin. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[17] J. H. et al. Zou. "SNiPER: an offline software framework for non-collider physics experiments". *J. Phys. Conf. Ser.* 664.7 (2015), p. 072053. DOI: 10.1088/1742-6596/664/7/072053.

[18] A. Compagnucci. *Development of Boosted Decision Trees for energy reconstruction of Inverse Beta Decay events in JUNO*. 2018.

[19] F. Vidaich. *Deep Neural Networks per la ricostruzione dell'energia di eventi di decadimento beta inverso nell'esperimento JUNO*. 2018.

[20] S-O Flyckt and M. Carole. *Photomultiplier Tubes - principles & applications*. An optional note. Photonis, Brive, France, July 2002.

[21] R. Sebastian. *An overview of gradient descent optimization algorithms*. 2016. eprint: arXiv:1609.04747.

[22] N. Perraudin et al. "DeepSphere: Efficient spherical Convolutional Neural Network with HEALPi sampling for cosmological applications". *Astronomy and Computing* (2018). arXiv: 1810.12186. URL: https://arxiv.org/abs/1810.12186.

[23] K. M. Górski et al. *The HEALPix Primer*. https://healpix.sourceforge.io/pdf/intro.pdf. 2018.

[24] A. Zonca et al. "healpy: equal area pixelization and spherical harmonics transforms for data on the sphere in Python". *Journal of Open Source Software* 4.35 (Mar. 2019), p. 1298. DOI: 10.21105/joss.01298. URL: https://doi.org/10.21105/joss.01298.

[25] Harislqbal88. *PlotNeuralNet*. https://github.com/HarisIqbal88/PlotNeuralNet. 2019.

[26] N. Perraudin et al. *Deepsphere: a spherical convolutional neural network*. https://github.com/SwissDataScienceCenter/DeepSphere. 2019.

[27] T. Birkenfeld. "Electron-Positron Discrimination in the Jiangmen Underground Neutrino Observatory with Deep Neural Networks". MA thesis. Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen, 2018.